**CSE 523/524**

Project name: Scatterplot evaluation metrics

Data of 828 scatterplots is available in csv format. 15 measures have been calculated on each scatterplot and these are available in csv format. Scores assigned by a human are also available.

My Ideas:

Calculate the 2002 metrics as mentioned in the original paper. Feed these metrics to a Neural Network. Assume human-assigned scores as ground truth. Try different combinations of these metrics and compare all results to see which combination yields best results. Try assigning weights to the important metrics, like the 15 best measures. Assign high weight to DSC as it is shown to be a very good metric. Compare accuracies to check which combination of metrics and weights yields good results.

I am now wondering if it were better to use support vector regression for this. Python implements this here <https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html>

Here’s a paper that has the details on (see section 1.2 for the basic idea)

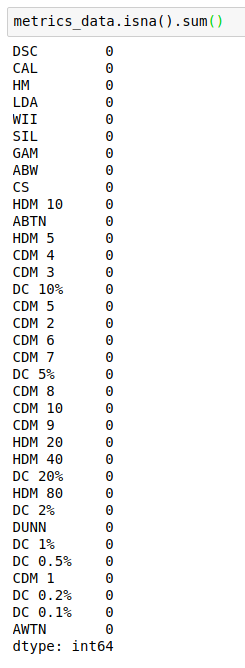
<https://alex.smola.org/papers/2003/SmoSch03b.pdf>

We can feed the user rating as the dependent variable and all the metrics as the dependent variables. It will learn the epsilon surface of the metric space. Once the surface has been learned any new scatterplot human judgement can be predicted.

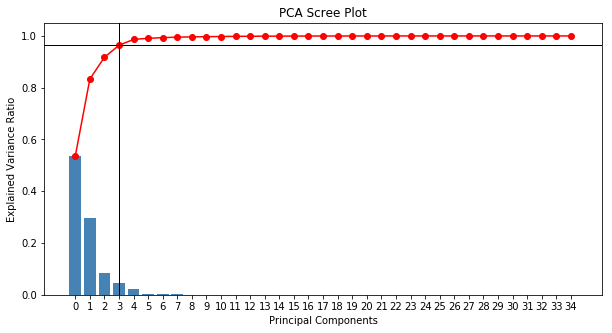
You can also research the topic of how to use neural networks for regression tasks, linear and nonlinear. Essentially, you wish to predict a value given the metric values as input. You would train it using the data you observed (the rows in the spreadsheet).

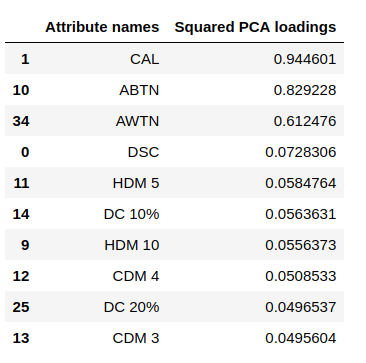
**Data cleaning:**

Data had no missing values or any other problems and thus didn’t require any cleaning.

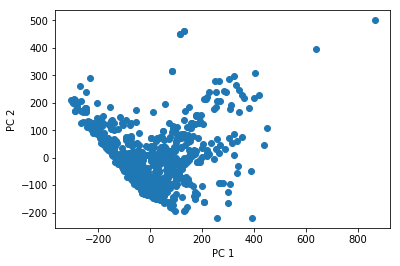


**PCA:**

* The dataset contains data about several metrics for scatterplots. I tried to find out which metrics explain the most variance in the data using PCA.
* The PCA scree plot is shown below. We see that the elbow of the curve is at n\_components=4. The first two components themselves explain more than 80% of the variance in the data.
* Further, I calculated the factor loadings of the attributes in order to find out which attributes make the most sense. The top 10 attributes according to the loadings are as follows:

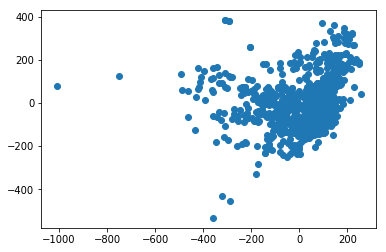


* DSC, which is known to be a good metric, was expected to show up on this list and it did. However, I also came to know that there are metrics better than DSC.
* Further, I projected all the data points onto the top two principal components and visualized a biplot:



* This shows a kinda inverse relationship between the projections on PC 1 and PC 2. This might come in useful in the future.

**MDS:**

* I performed MDS clustering in order to see if it would reveal anything interesting. The following is the graph of MDS clustering using Euclidean distance.
* We see that almost all the points are huddled together in one cluster. We can point out the outliers here. In the future, I guess that removing these outliers may yield better results.

Lastly, I have tried out a few models on the data. They are:

* Logistic Regression:
  + Accuracy: 0.5180722891566265
* Neural network:
  + Layers: 3 (35, 35 and 1 neurons each)
  + Accuracy: 0.463855421686747
* Support Vector Regression:
  + Accuracy: 0.5120481927710844

All code is available here: <https://github.com/AmitD26/CSE-523-Scatterplot-Separability-Metrics/blob/master/CSE%20523%20-%20Scatterplot%20separation%20measures.ipynb>

5/15: The task is to predict the human score. Can you use the models you learned above for prediction? You could train on ⅔ of the data and test with the other 1/3. You can try different such sets.

Yes. I have used train\_test\_split in python with test size as 0.2. Meaning that 80% of the data is used for training while 20% is used for testing. This can be changed as required.

I trained on ⅔ of the data and tested with the other ⅓ as you said. The new accuracies for that are as follows:

* Logistic Regression:
  + Accuracy: 0.511
* Neural network:
  + Layers: 3 (35, 35 and 1 neurons each)
  + Accuracy: 0.515
* Support Vector Regression:
  + Accuracy: 0.518

I tried changing the parameters of logistic regression in Python. Logistic regression has parameters like penalty (L1, L2, etc), class weight, solver, etc. The default class weights yielded an accuracy of 0.511. I tried changing the class weights and found out that this combination - {0: 1, 1: 6} improved the accuracy to 0.555. A “liblinear” solver is used with penalty L2.

Similarly, I tried to find a good set of parameters for support vector regression. It has mainly three parameters - C, gamma and epsilon. After tweaking these parameters, I found out that C = 10, gamma = 0.0001 and epsilon = 0.1 shows an improvement in the accuracy. The accuracy is 0.544. I used the default kernel (RBF).

Any suggestions about what I should do further? Thanks.